

# An Object-Oriented Parallel Particle-In-Cell Code for Beam Dynamics Simulation in Linear Accelerators

Ji Qiang and Robert D. Ryne  
*Ms H817, LANSCE-1*  
*Los Alamos National Laboratory*  
*Los Alamos, NM 87545*  
*Email: jiqiang@lanl.gov, ryne@lanl.gov*  
*Fax: 505-665-2906*

Salman Habib  
*Ms B288, T-8*  
*Los Alamos National Laboratory*  
*Los Alamos, NM 87545*  
*Email: habib@lanl.gov*

Viktor Decyk  
*Physics Department,*  
*University of California at Los Angeles*  
*Los Angeles, CA 90024*  
*Email: vdecyk@pepper.physics.ucla.edu*

SUBJECT CLASSIFICATIONS: 77G05 and 77C05

KEY WORDS: object-oriented, particle-in-cell, beam dynamics, linear accelerators

# Abstract

In this paper, we present an object-oriented three-dimensional parallel particle-in-cell code for beam dynamics simulation in linear accelerators. A two-dimensional parallel domain decomposition approach is employed within a message passing programming paradigm along with a dynamic load balancing scheme. An important feature of this code is the use of split-operator methods to integrate single-particle magnetic optics techniques with parallel Particle-In-Cell (PIC) techniques. By choosing a splitting scheme that separates the self-fields from the complicated externally applied fields, we are able to utilize a large step size and still retain high accuracy. The method employed is symplectic, and can be generalized to arbitrarily high order accuracy if desired. Performance tests on an SGI/Cray T3E-900 and an SGI Origin 2000 show good scalability of the object-oriented code. We present, as an example, a simulation of high current beam transport in the Accelerator Production of Tritium (APT) linac design.

## I. INTRODUCTION

The analysis and simulation of charged particle beam transport is an important subject in accelerator design and operation. The increasing interest in high intensity beams for future accelerator applications presents challenging problems that require one to understand and predict the dynamics of beams subject to complicated external focusing and accelerating fields, as well as the self-fields due to Coulomb interaction of the particles. One approach to studying the behavior of these particles is to use envelope equations [1–3]. These equations are a set of ordinary differential equations for the second order moments of the particle distribution and can be calculated quickly. However, the envelope equations do not provide a detailed description of the beam, and furthermore are not self-contained. Future accelerator applications put extremely stringent requirements on particle loss, which is associated with

the low density, large amplitude halo of the beam. The need to model the details of the beam distribution, in the presence of strong self-fields, leads us to a full Poisson-Vlasov description to better understand and predict the behavior of intense charged particle beams in accelerators.

The Poisson-Vlasov equations can be solved using a phase space grid-based method or a PIC method. The grid-based method is effective in one and two dimensions [4], but for three-dimensional systems with six phase space variables, the grid-based method will require an enormous amount of memory even for a coarse grid. Also, grid based methods may break down when very small-scale structures form in the phase space. The PIC method has a much lower storage requirement and will not break down even when the phase space structure falls below the grid resolution. This method is widely used to study the dynamics of high intensity beams in accelerators [5–7]. At present, most particle-based accelerator simulation codes run only on serial computers. The computational time cost associated with using a large number of numerical particles restricts that number, and limits the accuracy of the PIC calculation. Parallelism can significantly improve the accuracy of PIC simulation by enabling the use of a larger number of particles and a finer grid resolution. It also dramatically reduces the computation time. In this paper, we present an object-oriented parallel PIC code for beam dynamics simulation in linear accelerators. The use of parallel computing provides for high performance and high accuracy, while the object-oriented approach gives the program good maintainability, reusability, and extensibility. In addition to describing the object-oriented implementation on parallel computers, we will also describe the use of split-operator methods, which provide a powerful means to include space charge effects in single-particle beam transport codes. The result is a multi-particle capability that combines sophisticated techniques of magnetic optics with those of parallel PIC simulation.

The organization of this paper is the following: The physical model and numerical methods are described in Section 2. The parallel PIC algorithm using MPI on distributed parallel machines is discussed in Section 3. The object-oriented software design for beam dynamics simulation is given in Section 4. Performance tests are given in Section 5. An application of

the code to a simulation of the APT linac design is presented in Section 6. The conclusions are drawn in Section 7.

## II. PHYSICAL MODEL AND NUMERICAL METHODS

The equations governing the motion of individual particles in an accelerator (in the absence of radiation) are Hamilton's Equations,

$$\frac{d\vec{q}}{dt} = \frac{\partial \vec{H}}{\partial \vec{p}}, \quad \frac{d\vec{p}}{dt} = -\frac{\partial \vec{H}}{\partial \vec{q}}, \quad (1)$$

where  $H(\vec{q}, \vec{p}, t)$  denotes the Hamiltonian of the system, and where  $\vec{q}$  and  $\vec{p}$  denote canonical coordinates and momenta, respectively. Let  $\zeta$  denote the six-vector of coordinates and momenta. In the language of *mappings*, which are a major theme in modern accelerator physics, we would say that there is a (generally nonlinear) map,  $\mathcal{M}$ , corresponding to the Hamiltonian  $H$ , which maps initial phase space variables,  $\zeta^i$ , into final variables,  $\zeta^f$ , and we write

$$\zeta^f = \mathcal{M}\zeta^i. \quad (2)$$

Sophisticated techniques now exist to compute maps corresponding to externally applied electromagnetic fields to essentially any order, to combine and manipulate maps, to apply maps to phase space coordinates (or functions of the coordinates), and to analyze maps [8]. Note that Hamilton's equations can be rewritten as

$$\frac{d\zeta}{dt} = -[H, \zeta], \quad (3)$$

where  $[,]$  denotes the Poisson bracket. The corresponding equation governing the beam distribution function,  $f(\zeta, t)$ , is simply the Liouville equation,

$$\frac{df}{dt} = \frac{\partial f}{\partial t} - [H, f] = 0, \quad (4)$$

from which we obtain

$$\frac{\partial f}{\partial t} = [H, f]. \quad (5)$$

It is straightforward to show that the evolution of a distribution function (i.e. the solution of Eq. (5)) is also contained in  $\mathcal{M}$ . Namely, a distribution function  $f(\zeta, t)$  whose initial value is  $f^0(\zeta) = f(\zeta, 0)$  evolves according to

$$f(\zeta, t) = f^0(\mathcal{M}^{-1}\zeta). \quad (6)$$

So far we have implicitly assumed that we are dealing with particles subject only to externally applied fields. We can treat the dynamics in the presence of external fields and self fields (i.e. space charge fields) and including them in the single-particle Hamiltonian. In many cases one can write

$$H = H_{ext} + H_{sc}, \quad (7)$$

where  $H_{ext}$  denotes the Hamiltonian in the absence of self-fields and  $H_{sc}$  denotes the Hamiltonian associated with the space-charge fields. In accelerator physics,  $H_{ext}$  is often an extremely complicated function, perhaps containing hundreds of thousands of terms. This is due to the fact that it normally involves Taylor expansion of the Hamiltonian in order to perform high order perturbation theory around a reference trajectory. In contrast with the treatment of external fields, self-fields governed by  $H_{sc}$  are not normally treated as a power series in the canonical variables because the variation is too great over the domain of interest. In many cases  $H_{sc}$  is simply proportional to the scalar potential  $\phi$ , which satisfies the Poisson equation:

$$\nabla^2 \phi(\vec{q}) = -\rho(\vec{q}). \quad (8)$$

The combination of Eq. 5 and Eq. 8 constitutes the Poisson-Vlasov system of equations.

Our approach to solving these equations involves the use of split-operator methods. As just mentioned, beam dynamics calculations often involve a Hamiltonian that can be written as a sum of two parts,  $H = H_{ext} + H_{sc}$ . Such a form is ideally suited for the application

of symplectic split-operator methods [9]. More generally, consider a Hamiltonian that can be written as a sum of two parts  $H = H_1 + H_2$ , where each part, separately, can be solved exactly or to some desired accuracy or order. In other words, suppose that we can compute the mapping  $\mathcal{M}_1$  corresponding to  $H_1$  and the mapping  $\mathcal{M}_2$  corresponding to  $H_2$ . In our case  $H_1$  includes the external fields, and  $\mathcal{M}_1$  can be computed to any order using the techniques of Magnetic Optics; the second term,  $H_2$ , corresponds to the space-charge fields, and can be dealt with using parallel particle simulation techniques. Given  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , the following algorithm is accurate through 2nd order in  $\tau$ ,

$$\mathcal{M}(\tau) = \mathcal{M}_1(\tau/2) \mathcal{M}_2(\tau) \mathcal{M}_1(\tau/2) , \quad (9)$$

where  $\tau$  denotes the time step. (In accelerator physics, one often uses a coordinate as the independent variable. However, for the sake of this discussion we will continue to refer to  $\tau$  as a time step). As a side note, if we were to use a different splitting in which  $H_1$  depends only on momenta and  $H_2$  depends only on positions, then this algorithm is the same as the well-known leap-frog algorithm. However, the split-operator approach provides a powerful framework capable of dealing with the far more complicated Hamiltonians often encountered in accelerator physics. The method is easily generalized to more terms (i.e. more splittings) if necessary. Furthermore, symplectic split-operator methods are easily generalized to higher order accuracy in time. A well-known fourth-order algorithm is due to Forest and Ruth, and an arbitrary order scheme was derived by Yoshida [10] [11]. There are also implicit symplectic methods based on this approach that do not require the Hamiltonian to be split into a sum of exactly solvable pieces [9].

If we treat  $\mathcal{M}_1$  as corresponding to the external fields and  $\mathcal{M}_2$  as corresponding to the space-charge fields, Eq. 9 describes an algorithm to treat both single-particle magnetic optics effects and space charge effects. A time step involves the following: (1) transport of a numerical distribution of particles through half a step based on  $\mathcal{M}_{ext}$ , (2) solving Poisson's equation based on the particle positions and performing a space-charge "kick" (i.e. an instantaneous change in momenta, since  $H_{sc}$  depends only on coordinates, hence  $\mathcal{M}_{sc}$  only

effects momenta), and (3) performing transport through the remaining half of the step based on  $\mathcal{M}_{ext}$ . If the space charge is intense, this can be performed repeatedly on successive pieces of a beamline element; if the space charge is weak, it may be possible to achieve good accuracy by computing the space charge kicks infrequently, in which case  $\mathcal{M}_{ext}$  would correspond to a string of elements within a half-step. Thus, an important feature of this approach is that it enables one to use large time steps (i.e. large steps in the independent variable) in the regime of weak or moderate space charge. Essentially, it enables one to decouple the rapid variation of the externally applied fields from the more slowly varying space charge fields. If more accuracy is required, one can use the 4th order method of Forest and Ruth, although this requires 3 space charge calculations per full step instead of just one, and since this dominates the execution time it is costly.

Finally, a subtle point is that, while most beam dynamics codes use a coordinate as the independent variable (typically the longitudinal coordinate,  $z$ , in a linac code), Poisson's equation has to be solved *at fixed time*. Thus, prior to every space charge calculation it is necessary to convert from the canonical coordinates and momenta currently in use to the more usual coordinates and momenta in which  $(x, y, z)$  are known at fixed time. Such a calculation makes sense and is easily accomplished if the particle motion is essentially ballistic over a distance corresponding to the bunch length, since a bunch contains a distribution of arrival times, and they must all be moved to a fixed time.

In summary, split-operator methods provide the “glue” to join two major fields, Magnetic Optics and parallel particle simulation techniques. All that is required is (1) the ability to compute maps corresponding to external fields, (2) the ability to compute the space charge fields (normally accomplished using a parallel Poisson solver), and (3) a knowledge of the particle positions at fixed times, or the ability to compute it.

The physical system for beam dynamics studies consists of the beam and the accelerating/transport system which in turn contains a number of accelerating and focusing elements. These elements consist of drift spaces, magnetic quadrupoles, and RF accelerating gaps. The Hamiltonians corresponding to these elements, in the linear approximation, which we will

denote by  $K_{ext}$ , are as follows.

For the drift tube,  $K_{ext}$  is [12]

$$K_{ext} = \frac{1}{2}(P_x^2 + P_y^2) + \frac{1}{2\gamma_0^2\beta_0^2}P_t^2 \quad (10)$$

For the magnetic quadrupole,  $K_{ext}$  is [12]

$$K_{ext} = \frac{1}{2}(P_x^2 + P_y^2) + \frac{1}{2}k(z)(X^2 - Y^2) + \frac{1}{2\gamma_0^2\beta_0^2}P_t^2 \quad (11)$$

where the focusing strength,  $k(z)$ , is related to the quadrupole gradient according to

$$k(z) = \frac{q}{p^0}g(z) \quad (12)$$

For the accelerating RF gap,  $K_{ext}$  is [13]

$$\begin{aligned} K_{ext} = & \frac{\delta}{2lp^0}(P_x^2 + P_y^2) + \frac{l}{2\delta}\left[\frac{1}{p^0}\left(\frac{q}{2\omega}e'\sin\phi_s\right)^2 - \frac{q}{2\omega}(e'' + \frac{w^2}{c^2}e)\sin\phi_s\right](X^2 + Y^2) \\ & - \frac{qe'\sin\phi_s}{2p^0\omega}(XP_x + YP_y) + \frac{m^2\omega^2l\delta}{2(p^0)^3}P_t^2 - \frac{\omega qe\sin\phi_s}{2\omega^2l\delta}T^2 \end{aligned} \quad (13)$$

where  $e$  is the electric field,  $\phi_s$  is given by  $\phi_s = \omega t^g + \theta$ , and  $p^0$  is the design momentum.

In order to compute  $\mathcal{M}_{ext}$  for the RF gap, one needs to numerically solve the equations of motion for the design trajectory inside the gap. These equations are given by

$$(t^g)' = \frac{-p_t^g/c}{\sqrt{(p_t^g)^2 - m^2c^4}} \quad (14)$$

$$(p_t^g)' = -qe(z)\cos(\omega t^g + \theta) \quad (15)$$

Lastly, in addition to the Hamiltonians for the various external elements, we also need the Hamiltonian corresponding to the space charge field. This is given by

$$K_{self} = \frac{q/\delta c}{l\beta g(\gamma^g)^2}\phi \quad (16)$$

which includes electrostatic fields and azimuthal magnetic fields. The potential  $\phi$  can be obtained by convolving the charge density with a Green's function. In our code, the charge density is obtained by depositing the particles onto a grid using a cloud-in-cell (CIC) scheme.

The potential is expressed as

$$\phi_{p,q,r} = \sum G_{p-p',q-q',r-r'} \rho_{p-p',q-q',r-r'} \quad (17)$$

where  $G$  is the Green's function on the grid, and  $\rho$  is the charge density on the grid. Often, the beam size is much smaller than the inside wall radius of the accelerator, in which case we may treat the beam as an isolated system. In such a case, the above convolution can be calculated using a Fast Fourier Transform (FFT) technique given by Hockney [14].

### III. PARALLEL PARTICLE-IN-CELL ALGORITHM

A message passing programming paradigm with MPI (Message Passing Interface) is employed in our parallel particle-in-cell simulation. MPI is a standard library of message passing programming bound to C (C++) and Fortran [15]. In this paradigm, a computer program creates one or more processes. Each process can execute the same program or different program with local data. In most implementations, each process is mapped to a physical processor with a unique identification number. When the data from more than one processor is required, explicit communication is performed by calling library routines to send or receive messages from other processors. Hence, in this programming model, the programmer has to control the data distribution on the processors and communication among processors. This gives it the advantages of flexibility and better performance. However, this also increases the difficulty of parallel programming. Applying object-oriented design to parallel message passing programming helps to encapsulate the details of communication and data distribution. This enables the user to manage the applications at a higher level.

A two-dimensional domain-decomposition approach is employed in the parallel particle simulation [16,17]. A schematic plot of the two-dimensional decomposition on the y-z plane is shown in Fig. 1. The solid grid lines define the computational domain grids. The dashed lines define the local computational domain on each processor. Here, the boundary grids are the out-most grids inside the physical boundary. The guard grids are used as temporary storage of grid quantities from the neighboring processors. The physical computational domain is defined as a 3-dimensional rectangular box with range  $x_{min} \leq x \leq x_{max}$ ,  $y_{min} \leq y \leq y_{max}$ ,

and  $z_{min} \leq z \leq z_{max}$ . This domain is decomposed on the  $y - z$  plane into a number of small rectangular blocks. These blocks are mapped to a logical two-dimensional Cartesian processor grid. Each processor contains one rectangular block domain. The range of a block on a single processor is defined as  $x_{min} \leq x \leq x_{max}$ ,  $y_{lmin} \leq y \leq y_{lmax}$ , and  $z_{lmin} \leq z \leq z_{lmax}$ . Here, the subscript *lmin* and *lmax* specify local minimum and local maximum. The mesh grid is defined to store the field-related quantities such as charge density and electric field. The number of grid points along three dimensions on a single processor is defined as:

$$Nx_{local} = \text{int}[(x_{max} - x_{min})/hx] + 1 \quad (18)$$

$$Ny_{local} = \text{int}[(y_{lmax} - y_{min})/hy] - \text{int}[(y_{lmin} - y_{min})/hy] + N_g \quad (19)$$

$$Nz_{local} = \text{int}[(z_{lmax} - z_{min})/hz] - \text{int}[(z_{lmin} - z_{min})/hz] + N_g \quad (20)$$

where  $hx$ ,  $hy$ , and  $hz$  are the mesh sizes along  $x$ ,  $y$  and  $z$  direction respectively. The quantity  $N_g$  refers to the number of guard grids in  $Ny_{local}$  and  $Nz_{local}$ .  $N_g = 2$  if the number of processors in that dimension is greater than 1; otherwise,  $N_g = 1$ . For the processor containing the starting grid in the global mesh, there is one more grid point along the  $y$  and  $z$  directions. The particles with spatial positions within the local computational boundary are assigned to the processor containing that part of physical domain.

The parallel computation starts with constructing a 2-D logical Cartesian processor grid, reading input data from processor 0 and broadcasting it to the other processors, setting up the local initial computational domain, initializing objects, and generating particles from the initial distribution function. There are three approaches to generating the particles local to the processor at the beginning of the simulation. One approach is that each processor generates the average number of particles by sampling the whole initial distribution. Then explicit all-to-all communication is used to send the particles to the appropriate processor where it locates. This approach has the advantage that each processor needs only to generate small number of particles. However, the communication costs will increase with the increasing number of processors and particles, which makes this approach less scalable. The second

method is that each processor generates the total number of numerical particles of initial distribution. Only particles local to the processor are kept, and the other particles not local to the computational domain are thrown away. This approach avoids the communication cost and uses the same amount of time with increasing number of processors. Nevertheless, this approach is extremely inefficient, and the time cost is the same regardless of the number of processors used. The third approach is that each processor generates the average number of numerical particles by sampling a part of initial distribution using a rejection method [18]. This part of the initial distribution contains the computational domain local to the processor. Hence, the particles generated from this distribution will be local to the processor. There is no need for communication. This approach has the advantage of scaling with increasing number of processors. The disadvantage of this approach is that the result may not be reproducible using a different number of processors partly due to the difference in random number generation on each processor.

The particles generated on each processor will advance following the maps defined in Section 2. If a particle moves outside the local computational domain, it will be sent to the corresponding processor where it is located. A particle manager function is defined to handle the explicit communication using MPI among two-dimensional processor grids. The  $y$  and  $z$  positions of every particle on each processor are checked. The particle is copied to one of its four buffers and sent to one of its four neighboring processors when its  $y$  or  $z$  position is outside the local computational domain. After a processor receives the particles from its neighboring processors, it will decide among those particles whether some of them will be further sent out or not. The outgoing particles are counted and copied into four temporary arrays. The remaining particles are copied into another temporary array. This process is repeated until there is no outgoing particle on all processors to be found. Then, the particles in the temporary storage along with the particles left in the original particle array are copied into a new particle array.

After each particle moves to its local computational domain, a linear CIC particle-deposition scheme is done for all processors to get the charge density on the grid. For

the particles located between the boundary grid and computational domain boundary, these particles will also contribute to the charge density on the boundary grids of neighboring processors. Hence, explicit communication is required to send the charge density on the guard grids, which is from the local particle deposition, to the boundary grids of neighboring processors to sum up the total charge density on the boundary grids. With the charge density on the grids, Hockney's FFT algorithm is used to solve the Poisson's equation with open boundary conditions. Due to this algorithm, the original grid number is doubled in each dimension. The charge density on the original grids is kept the same. The charge density on the other grids is set to 0. The Green's function is defined as

$$G_{p,q,r} = \frac{1}{\sqrt{(hx(p-1))^2 + (hy(q-1))^2 + (hz(r-1))^2}}, \quad (21)$$

where  $p = 1, \dots, Nx_{local}^* + 1$ ,  $q = 1, \dots, Ny_{local}^* + 1$ ,  $r = 1, \dots, Nz_{local}^* + 1$ . Here,  $Nx_{local}^*$ ,  $Ny_{local}^*$ ,  $Nz_{local}^*$  are the local computation grid number without including guard grids in all three dimensions. For the grid points outside the above boundary, symmetry is used to define the Green's function on these grids.

$$G_{p,q,r} = G_{2Nx-p+2,q,r}, \quad p = Nx_{local}^* + 2, 2Nx \quad (22)$$

$$G_{p,q,r} = G_{p,2Ny-q+2,r}, \quad q = Ny_{local}^* + 2, 2Ny \quad (23)$$

$$G_{p,q,r} = G_{p,q,2Nz-r+2}, \quad r = Nz_{local}^* + 2, 2Nz \quad (24)$$

Communication is required to double the original distributed 3-dimensional grid explicitly. This can be avoided by including this process into the 3-dimensional FFT. In the 3-dimensional parallel FFT, we have taken advantage of the undistributed dimension along the  $x$  dimension, where a local serial FFT can be done in that dimension for all processors. A local temporary two-dimensional array with size  $(2Nx, Ny_{local})$  is defined to contain part of the charge density at fixed  $z$ . The charge density on the original grid is copied into the  $(Nx, Ny_{local})$  part of the temporary array. The rest of the temporary array is filled with 0. In the case of the FFT of the Green's function, symmetry can be used to obtain the values of the Green's function in the region  $(Nx + 2, Ny_{local})$ . After the local two-dimensional

FFT along  $x$  is done, it is copied back to a slice of a new 3-dimensional array with size  $(2Nx, Ny_{local}, Nz_{local})$ . A loop through  $Nz_{local}$  gives the FFT along  $x$  for the three dimensional array. Then, a transpose is used to switch the  $x$  and  $y$  indices. Now, the 3-dimensional matrix has size  $(Ny, Nx'_{local}, Nz_{local})$ . Here,  $Nx'_{local}$  is the new local number of grids in the  $x$  dimension along the  $y$  dimension processors. A similar process is done to obtain the FFT along the  $y$  direction for double-size grids  $(2Ny, Nx'_{local}, Nz_{local})$ . Another transpose is used to switch the  $y$  and  $z$  indices and a local FFT along  $z$  with a double-size grid is done on all processors to finish the 3-dimensional FFT for the double-size grid in all three dimensions. During the inverse parallel FFT, a reverse process is employed to obtain the potential on the original grids. In the transpose of indices, global all-to-all communication is used.

From the potential on the grid, we calculate the electric field on the grid using a central finite difference scheme. To calculate the electric field on the boundary grid, the potential on the boundary grid of neighboring processors is required. A communication pattern similar to the one used in the charge density summation on the boundary grids is used to send the potential from the boundary grids to the guard grids of neighboring processors. After the electric field on the grids is obtained, it has to be interpolated from the grids onto the local particles to push the particles. Since we have used the linear CIC scheme, the electric field of particles between the boundary grid and computational domain boundary will also depend on the electric field on the boundary grid of neighboring processors. A similar communication pattern is used to send the electric field from the boundary grids to the guard grids of the neighboring processors. With the electric field on grids local to each processor, the interpolation is done for all processors to obtain the space charge force on every particle. The local particles are updated in momentum space based on the space charge force. This operation defines the mapping  $\mathcal{M}_2$ .

Dynamic load balancing is employed with adjustable frequency to keep the number of particles on each processor approximately equal. A density function is defined to find the local computational domain boundary so that the number of particles on each processor is roughly balanced. This number depends on the local integration of the charge density

on each processor. To determine the local boundary, first, the three-dimensional charge density is summed up along the  $x$  direction on each processor to obtain a two-dimensional density function. This function is distributed locally among all processors. Then, the two-dimensional density function is summed up along the  $y$  direction to get the local one-dimensional charge density function along  $z$ . This density function is broadcast to the processors along the  $y$  direction. The local charge density function is gathered along  $z$  and broadcast to processors along the  $z$  direction to get a global  $z$  direction charge density distribution function on each processor. Using this global  $z$  direction density distribution, the local computational boundary in the  $z$  dimension can be determined assuming that each processor contains a fraction of total number of particles about equal to  $1/nproc_z$ . Here,  $nproc_z$  is the number of processors along the  $z$  direction in the two-dimensional Cartesian processor grid. A similar process is used to determine the local computational boundary in the  $y$  direction. Strictly speaking, the above algorithm will work correctly for a two-dimensional density distribution function which can be separated as a product of two one-dimensional functions along each direction. However, from our experience, this algorithm works reasonably well in beam dynamics simulation in the linear accelerator.

#### IV. OBJECT-ORIENTED SOFTWARE DESIGN

The above parallel particle-in-cell algorithm is implemented in an object-oriented framework for the accelerator simulation. Object-oriented software design is a method of design encompassing the process of object-oriented decomposition [19]. After analysis of the (complex) physical system, the system is first decomposed into simpler physical modules. Next, objects are identified inside each module. Then, classes are abstracted from these objects. Each class has interfaces to communicate with the outside environment. Relationships are then built up among different classes and objects. These classes and objects are implemented in a concrete language representation. The implemented classes and objects are tested separately and then put into the physical module. Each module is tested separately

before it is assembled into the whole program. Finally, the whole program is tested to meet the requirements of problem.

Our application of the above-mentioned object-oriented design methodology to beam dynamics studies in accelerators results in the decomposition of the physical system into five modules. The first module handles the particle information consisting of the *Beam*, *BeamBC*, and the *Distribution* classes. The second module handles information regarding quantities defined on the field grid containing *Field* and *FieldBC* classes. The third module handles the external focusing and accelerating elements containing the *BeamLineElem* base class and its derived classes, the drift tube class, the quadrupole classes, and the rf gap class. The fourth module handles the computational domain geometry containing the *Geometry* class. The last module provides auxiliary and low level classes to handle explicit communication and input-output containing the *Pgrid2d*, *Communication*, *Utility*, *InOut* and *Timer* classes. The class diagram of the object-oriented model for a beam dynamics system is presented in Fig. 2. Here, run-time polymorphism is used to implement different external beam line elements. A single operation using the function of the beam-line-element base class can automatically select the appropriate function from different concrete beam-line-element class objects to execute [20]. The *inheritance* relation in Fig. 2 defines an “is” relationship among classes. The *aggregation* defines a relation that a class has an object of another class in its data member. The *use* defines a relation that a class uses an object of another class in its member function. The above object-oriented design is implemented using both Fortran 90 and the POOMA C++ framework [21]. In this paper, we only show the simulation results using the F90/MPI code.

## V. PERFORMANCE TEST

The performance of the object-oriented code was tested on both the SGI/Cray T3E-900 and the SGI Origin 2000. The SGI/Cray T3E-900 is a scalable, logically shared, physically distributed multi-processor machine with a range of configurations up to thousands of pro-

cessors [22]. Each node consists of a DEC Alpha 64-bit RISC microprocessor, local memory, system control chip and some network interfaces. The RISC microprocessor is cache-based, has pipelined functional units, and issues multiple instructions per cycle. The clock speed is 450 MHz. Each node has its own local DRAM memory with a capacity of from 64 Mbytes to 2 Gbytes. A shared, high-performance, globally addressable memory subsystem makes these memories accessible to every node. There are two-level on chip caches which can only be cached by local memory: one with 8 KB instruction and data caches and another with 96 KB 3-way associative cache. The nodes are connected by a high-bandwidth, low-latency bi-directional 3D torus interconnect network system.

The SGI Origin2000 is a scalable, distributed shared-memory multi-processor machine. It consists of a number of processing nodes linked together by a multi-dimension interconnection fabric. Each processing node contains either one or two processors, a portion of shared memory, which is physically distributed locally to each node but is also accessible to all other processors through the interconnection fabric. Each node also contains a directory for cache coherence, and two interfaces to connect to I/O devices and to link system nodes through the interconnection fabric. The processor used in the SGI Origin2000 is the MIPS R10000, a high-performance 64-bit superscalar processor with 4 GB memory, 32 KB on chip data cache, 32 KB on chip instruction cache, and 4 MB secondary cache. The single node clock speed for the system we used is 250 MHz. A cabinet can consist of up to 128 processor nodes [23].

The effect of dynamic load balancing is exhibited in Fig. 3. It shows the largest and smallest number of particles on one processor, as a function of time, with and without dynamic load balancing, on 16 processors. The total numerical particle number is 2 million with  $64 \times 64 \times 64$  grids. We see here that with the dynamic load balance the difference between the maximum number of particles and the minimum number of particles has been drastically reduced. This demonstrates that the dynamic load balance algorithm in our paper works well. In Fig. 4, we also give a comparison of the execution time on the Cray T3E as a function of number of processors using one-dimensional and two-dimensional parallel

processor partitions. In this simulation, we have used 2.6 million particles and  $64 \times 64 \times 64$  grids. The two-dimensional partition shows better scalability and is faster than the one-dimensional partition. This is because a two-dimensional processor partition has a more favorable surface-to-volume ratio. Communication cost is proportional to the surface area of the subdomain, whereas computation is proportional to its volume. Fig. 5 shows the time costs as a function of processor number on the SGI/Cray T3E-900 and on the SGI Origin 2000 for the same problem as in Fig. 4. Good scalability of our object-oriented parallel particle-in-cell code has been achieved. The lower execution time on the SGI Origin using 4 processors may be due to the larger cache size of this machine.

## VI. APPLICATION

As an application, we simulated the beam transport through three superconducting sections in a design of the APT linac [24]. The first section accelerates the beam from 211.4 MeV to 242.0 MeV, and contains six 2-cavity cryomodules. The second section accelerates the beam from 242.0 MeV to 471.40 MeV, and contains thirty 3-cavity cryomodules. The third section accelerates the beam to 1.03 GeV, and contains thirty five 4-cavity cryomodules. The major physical parameters in the design are listed in Table 1.

Energy gain:	211.4 - 1.03 GeV
Beam current:	0.1 A
Accelerator length:	513.58 m (includes three sections)
Quadrupole gradient:	5.60-5.10, 5.50-6.05, 5.00-7.25 T/m
Accelerating gradient:	4.30-4.54, 4.30-5.01, 5.246 MV/m
Synchronous phase:	-30 to -35, -30 to -42, -30 degree

The external focusing and accelerating fields for the first two cryomodules are given in Fig. 6. A quadrupole-doublet focusing lattice is used to provide transverse strong focusing and to reduce the focusing period comparing with a singlet lattice. The external longitudinal rf field is obtained from a MAFIA [25] calculation of the 5 cell superconducting cavity. For the above physical parameters and external field, we have performed the simulation using 20 million numerical particles on  $128 \times 128 \times 128$  grids. The initial distribution used here is

a six-dimensional Gaussian distribution in phase space. Fig.7 gives the transverse beam rms size and maximum amplitudes as a function of kinetic energy of beam. A jump in transverse rms beam size around 480 MeV is due to the jump in external focusing between the second section and the third section. The maximum transverse amplitudes set the lower bound of the minimum aperture that can be achieved in the design. Fig. 8 shows the longitudinal phase space plots at the end of the linac. The spiral structure suggests the formation of beam halo due to the mismatched focusing which can be understood using a simple particle-core model [3]. Particles in the beam halo will be lost if they move to large amplitude and strike the beam pipe.

## VII. CONCLUSIONS

In the above sections, we presented an object-oriented three-dimensional parallel particle-in-cell program for beam dynamics simulation in linear accelerators. This program employs a domain decomposition method with MPI. A dynamic load balance scheme is implemented in the code. It also has better maintainability, reusability, and extensibility compared with conventional structure based code. Performance tests on the SGI/Cray T3E-900 and the SGI Origin 2000 show good scalability. This code was successfully applied to the simulation of beam transport through three superconducting sections in the APT linac design.

## ACKNOWLEDGMENTS

We would like to thank Drs. Barbara Blind, Frank Krawczyk, and Thomas Wangler for RF superconducting data. This work was performed on the SGI/Cray T3E at the National Energy Research Scientific Computing Center located at Lawrence Berkeley National Laboratory, and the SGI Origin 2000 at the Advanced Computing Laboratory located at Los Alamos National Laboratory. This work was supported by the DOE Grand Challenge in Computational Accelerator Physics.

## REFERENCES

- [1] F. Sacherer, IEEE Trans. Nuc. Sci. NS-18 (1971) 1105.
- [2] J. Struckmeier and M. Reiser, Particle Accelerators 14 (1984) 227.
- [3] R. L. Gluckstern, Phys. Rev. Letters 73 (1994) 1247.
- [4] R. Ryne and S. Habib, in: Computational Accelerator Physics, ed. J. J. Bisognano and A. A. Mondelli, AIP Conference Proceedings 391, Woodbury, New York (1997) p. 377.
- [5] B. B. Godfrey, in Computer Applications in Plasma Science and Engineering, ed. by A. T. Drobot, Springer-Verlag, New York, 1991.
- [6] A. Friedman, D. P. Grote and I. Haber, Phys. Fluids B 4 (1992) 2203.
- [7] T. Wrangler, Principles of RF Linear Accelerators, John Wiley & Sons, New York, 1998.
- [8] A. J. Dragt, Particle Accelerators 55 (1996) 499.
- [9] E. Forest, et al., Phys. Lett. A 158 (1991) 99.
- [10] E. Forest and R. Ruth, Physica D43 (1990) 105.
- [11] H. Yoshida, Phys. Lett. A 150 (1990) 262.
- [12] R. D. Ryne, Computational Methods in Accelerator Physics, in preparation (1999).
- [13] R. D. Ryne, The linear map for an rf gap including acceleration, LANL rep. No.836 R5 ST 2629, (1991).
- [14] R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles, Adam Hilger, New York, (1988).
- [15] M. Snir, S. Otto, S. H. Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference, vol. 1, The MIT Press, Cambridge, MA, (1998).
- [16] P. C. Liewer and V. K. Decyk, J. Comp. Phys., 85, (1989) 302.

- [17] P. M. Lyster, P. C. Liewer, R. D. Ferraro, and V. K. Decyk, Computers in Physics, 9 (1995) 420.
- [18] M. H. Kalos and P. A. Whitlock, Monte Carlo Methods, John Wiley & Sons, New York , (1986).
- [19] G. Booch, Object-Oriented Analysis and Design with Applications, Benjamin/Cummings, Menlo Park, CA, (1994).
- [20] V. K. Decyk, C. D. Norton, and B. K.Szymanski, Scientific Programming, Vol. 6, Num. 4, IOS Press, (1997) p. 363.
- [21] W. Humphrey, R. Ryne, T. Cleland, et. al., in Computing in Object-Oriented Parallel Environments, ed. by D. Caromel, R. R. Oldehoeft, and M. Tholburn, Lecture Notes in Computer Science, vol. 1505, 1998.
- [22] <http://www.cray.com/products/systems/cray3e/overview.html>, (1998) .
- [23] <http://www.sgi.com/origin/2000/> (1998).
- [24] G. P. Lawrence, High-Power Proton Linac For APT: Status of Design and Development, in Proceeding of Linac98, Chicago, IL, 1998.
- [25] T. Weiland, Int. Journal of Numerical Modelling 9, 295-319 (1996)

## FIGURES

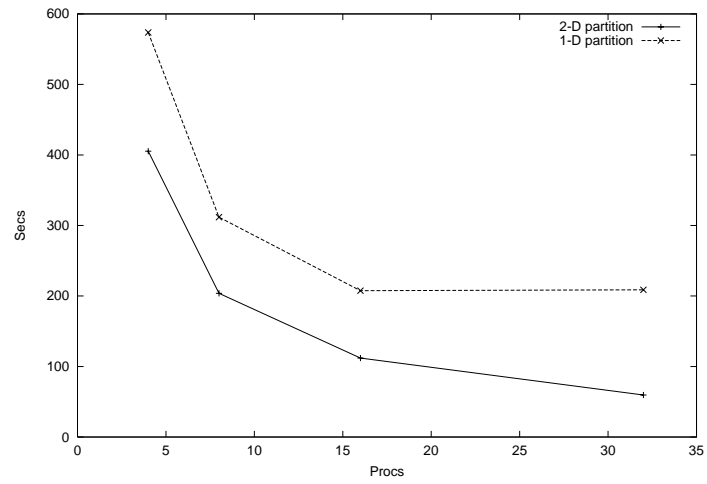


FIG. 1. A schematic plot of two-dimensional decomposition on y-z domain.